Outillage CLI : intégration aux systèmes d'entreprise

Famille de petits outils en ligne de commande qui parlent aux outils d'entreprise (gestionnaire de tickets, wiki, service de rendu de diagrammes), utilisables depuis Linux comme depuis Windows, et exposables à un assistant de code.

Fiche de méthode associée : `../../docs/06-cli-bi-plateforme.md`

Ce que ça fait

Outil	Rôle
`jira-update`	toutes les écritures sur un ticket : commentaires, statuts, champs, worklogs, liens, sprints, création, suppression
`confluence-publish`	création et mise à jour de pages de wiki, avec verrouillage optimiste et détection de mauvais format
`ticket-init`	amorçage d'un dossier de demande : structure, fichier de suivi prérempli depuis le gestionnaire de tickets
`drawio-to-image`	conversion d'un diagramme en image via un service de rendu local, pour qu'un assistant puisse le lire
`drawio-check-overlap`	détection de chevauchements de formes dans un diagramme
`wsl-write` / `wsl-append`	écriture d'un fichier côté Linux depuis Windows, contenu par entrée standard

Architecture

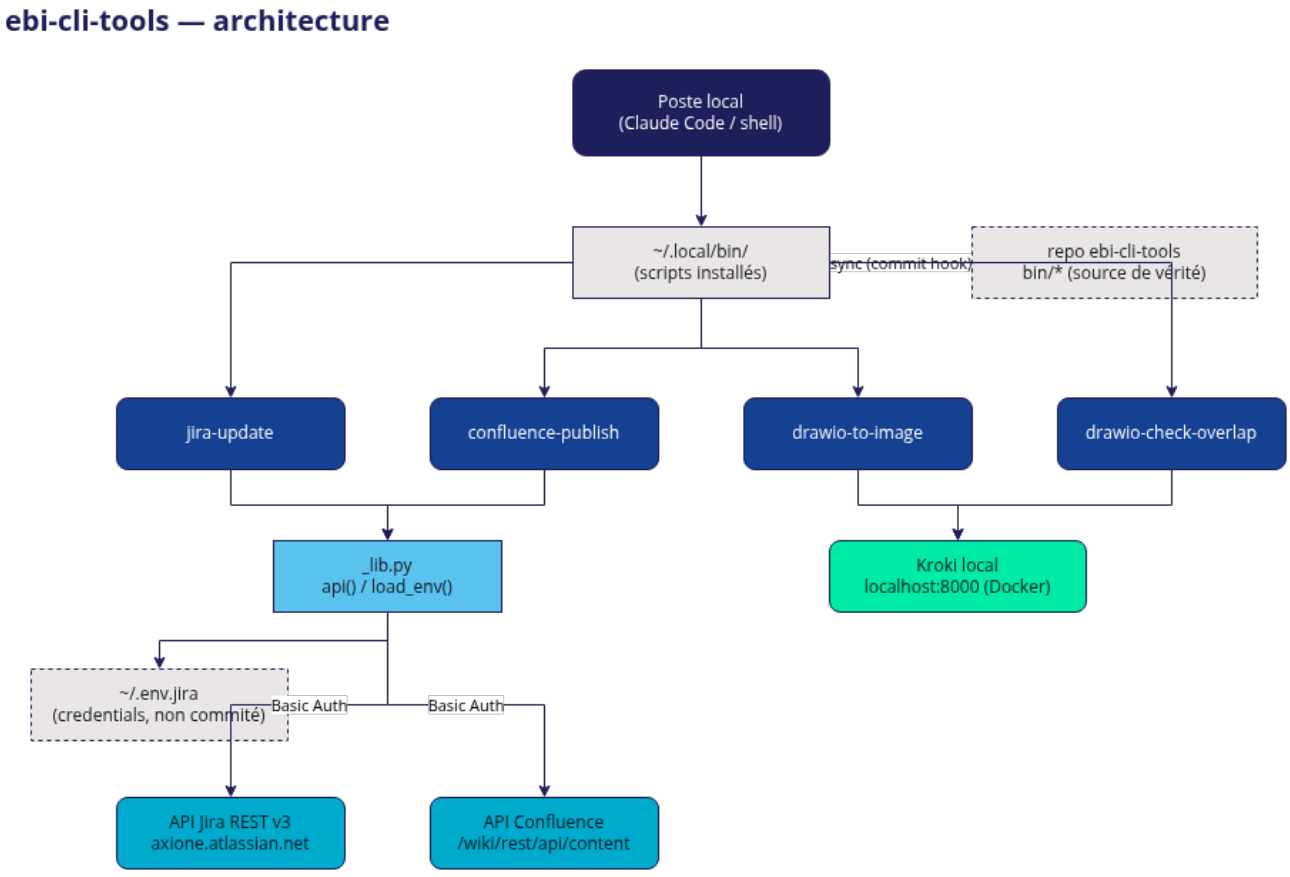


Figure 1: architecture

```
bin/                        source unique de vérité (Linux)
├── _lib.py                 bibliothèque partagée : configuration, authentification,
│                           point d'interception unique des appels d'API
├── jira-update             exécutables, sans extension
├── confluence-publish
└── ...

bin-windows/                couche d'appel Windows
├── wsl-run.py              pont générique unique
└── *.cmd                   trois lignes chacun, délèguent au pont
```

Ce qui vaut d'être regardé

1. Le point d'interception unique (`bin/_lib.py`, fonction `api`)

Le pattern le plus rentable du projet. Toutes les écritures passent par une seule fonction, qui porte l'authentification, le mode simulation et les garde-fous de contenu.

Conséquence : impossible d'oublier de gérer le mode simulation dans une nouvelle fonctionnalité. Tester le drapeau branche par branche fonctionne jusqu'à la septième branche, où quelqu'un oublie et écrit en production pendant un test. C'est arrivé, puis corrigé structurellement.

2. Le pont multi-plateforme (`bin-windows/wsl-run.py`)

Une seule enveloppe générique plutôt qu'un script dupliqué par outil. Trois points critiques, chacun issu d'un bug réel :

- conversion de chemins avec remplacement des antislashs avant l'appel à l'utilitaire de conversion, car la couche d'interopérabilité les consomme ;
- quotage de chaque argument, pas seulement ceux contenant une espace : c'est ce qui règle la troncature des textes accentués ;
- marque d'ordre des octets retirée aux deux extrémités, car un double marquage est possible.

3. La résolution dynamique (`bin/jira-update`, fonction `resolve_issue_type`)

Le référentiel de valeurs valides est interrogé côté serveur plutôt que codé en dur. En cas de valeur inconnue : échec explicite avec la liste des valeurs valides et une suggestion, jamais de repli silencieux.

Le repli silencieux créait des objets du mauvais type sans avertir. Un test de non-régression verrouille ce comportement.

4. La suggestion en deux passes (`bin/_lib.py`, fonction `suggest`)

Recherche par sous-chaîne d'abord (insensible aux accents), similarité approximative ensuite. La similarité seule est aveugle aux inclusions : une chaîne courte noyée dans un libellé long n'atteint jamais le seuil.

5. Les tests (`bin/test_lib.py`, `bin-windows/test_wsl_run.py`)

91 tests, bibliothèque standard uniquement. Points remarquables :

- chargement d'un exécutable sans extension pour le rendre testable, via un chargeur explicite ;
- test du point d'interception : l'écriture est bloquée en simulation, la lecture passe toujours ;
- tests de non-régression nommés par le bug qu'ils empêchent de revenir ;
- tests de quotage multi-plateforme exécutables sans avoir le sous-système Linux disponible.

6. Le journal de frictions (`FRICTIONS.md`, `FRICTIONS-ARCHIVE.md`)

Format en cinq champs (symptôme, contexte, cause, contournement, piste de correction), avec cycle de vie : le fichier actif ne garde que l'ouvert, l'archive conserve les entrées résolues avec leur résolution.

L'archive est devenue une base de connaissances des pièges d'API non documentés ailleurs. C'est souvent la partie la plus utile du dispositif au bout d'un an.

Fiche associée : `../../docs/17-frictions-amelioration-continue.md`

Vérification après anonymisation

Contrôle	Résultat
Compilation de tous les fichiers Python	ok
Syntaxe des scripts shell	ok
Tests unitaires (`bin/`)	73 passent

Tests du pont multi-plateforme (`bin-windows/`)	18 passent
Schémas régénérés depuis les sources anonymisées	4 images
Motifs sensibles résiduels	aucun

Ce qui a été retiré

- l'historique de contrôle de version (il contient ce qui a été supprimé) ;
- les caches d'outillage (analyse, tests, compilation) ;
- toute valeur réelle dans le fichier d'exemple de configuration : il ne porte que des noms de variables.

Les marqueurs d'anonymisation (`<COMPANY>`, `<INTERNAL_TRACKER_HOST>`, `<USER>`, `<TICKET>`…) remplacent les valeurs internes de façon cohérente : un même élément réel donne toujours le même marqueur.